

# COMP30027 A2 Report

Hannah Luo, Kenneth Lim

Main Body Word Count: 2629

## 1. Introduction

In COMP30027 Assignment 2, we were given the task of building a classifier to predict 43 Traffic Sign classes based on training data from the GTSRB (German Traffic Sign Recognition Benchmark) dataset.

In tackling this assignment, we experimented with 4 classic ML models: **Random Forest**, **KNN**, **SVM**, **Stacking** (RF + KNN + SVM); and 1 deep learning model: **CNN**. To train each of these models, we explored a range of ML techniques in the classification pipelines, including feature extraction / processing, data manipulation, and performed suitable hyper-parameter tuning to each model tailored to the training-evaluation phase.

In this report, we will present methodologies and results which followed each of these model training processes using numerical and visual reports. Furthermore, we will analyse the effectiveness of each method, discuss the strengths and limitations of our approaches, and offer insights into the underlying causes for each model's performance.

## 2. Methodology

This section outlines the pipeline used to construct and evaluate our classification models. Our approach consists of several key stages: data preprocessing, feature extraction, feature selection / enhancement, and model training. We used computer vision techniques combined with classic machine learning classifiers (RF, KNN, SVM, Stacking), complemented by deep learning model (CNN) for comparison.

### 2.1 Exploratory Data Analysis

We plot the distribution of class from training data (Figure 1). There is clear class imbalance.

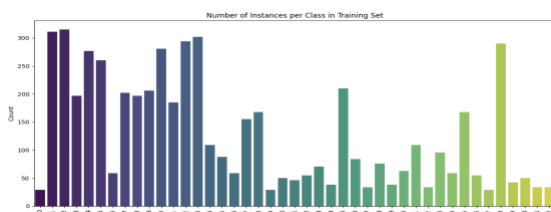


Figure 1- Distribution of classes in given data.

### 2.2 Data Augmentation (classic ML models)

A previous attempt of training the classifiers on the given image data only resulted in a very low 42% test accuracy (e.g. 57.367% for RF). We realised that this could be an issue to do with overfitting on training data.

To improve model generalisability and account for variations in lighting, orientation, and scale, we performed data augmentation (using the Albumentations library) on training images. The augmentation pipeline included random **rotations**, **scaling**, and **brightness contrast** (Figure 2). As a result, the training data size was doubled. The process increased the diversity of the training set for classifier training, and reduced model sensitivity to spatial orientation, colour and contrast.



Figure 2- Three examples of training images before and after data augmentation.

### 2.3 Data Augmentation (deep learning)

Similarly to Section 2.2, label-invariant transformations are applied to a copy of every image array. To do so, training images are first converted into grayscale, which reduces model complexity. Then, they are resized to 128x128 pixels, normalized and stored as a 3d array (height, width, number of channels = 1). To prevent data leakage, the images split into training and validation sets using random stratification. Data augmentation was applied to a copy of every image array, such as **random rotation**, **contrast**, **zoom**, **translation** and **gaussian noise**, then concatenated to train data.

## 2.4 Feature Extraction (classic ML models)

Though we were provided with train-ready features (color histograms, HOG, edge density etc.), we found the model performed poorly on those features alone (e.g. 75.057% mean cross-validation on RF). To capture a more texture and shape data, we extracted a diverse set of features from each image, outlined below. For these features, those with high dimensionality were PCA reduced to  $n$  components which retained 95% variance. All extracted features were concatenated to form a comprehensive feature vector for each image, resulting in 493 feature columns for each row.

**Local Binary Patterns (LBP)** is a pixel neighbourhood thresholding feature, chosen for its ability to capture texture pattern information. It makes the data robust to lighting changes and captures micro structural variations effectively but is sensitive to noise and rotation / scale variant. LBP features had 10 components.

**Color Histogram (CH)** found colour variance patterns in the RGB colour space, but it is ineffective due to augmented data and lighting variations. CH features were PCA reduced from 96 to 5 components for complexity reduction.

**Discrete Cosine Transform (DCT)** captures frequency components and overall pattern of the image but can lose spatial detail and has limited discriminative power. DCT features had 5 components.

**Hu Moments (HM)** is a set of seven moments chosen for its ability to capture global shape characteristics which are invariant to rotation, scale, and translation. However, it has low descriptive capacity and misses texture. HM features had 7 components.

**Gabor Filters (GF)** captures texture and edge information for distinguishing similar shapes (mimics human vision), but it is parameter-sensitive and computation heavy. GF features were PCA reduced from 80 to 9 components.

**Histogram of Oriented Gradients (HOG)** capture gradient structure and edge orientation for shape and structure recognition, but it is highly dimensional and alignment sensitive. HOG was PCA reduced from 1764 to 457 components.

## 2.5 Feature Selection (classic ML models)

Given the high dimensionality of the combined feature vector, we applied L1-regularised Logistic Regression (LASSO) with logistic regression for feature selection. This method promotes sparsity in the learned coefficients,

effectively eliminating less informative features and serving as a crucial step in training of classic ML models. Each C-value (strength of regularisation) generated top informative features (Table 1), providing valuable insights and thus improving computational efficiency and generalisation.

|        | Number of feature columns selected |    |     |    |    |     |
|--------|------------------------------------|----|-----|----|----|-----|
| C      | LBP                                | CH | DCT | HM | GF | HOG |
| 0.0020 | 0                                  | 0  | 0   | 0  | 0  | 11  |
| 0.0050 | 4                                  | 0  | 2   | 0  | 4  | 57  |
| 0.0075 | 5                                  | 0  | 2   | 0  | 6  | 96  |
| 0.0100 | 7                                  | 0  | 4   | 0  | 6  | 135 |
| 0.0125 | 7                                  | 0  | 5   | 2  | 9  | 167 |
| 0.0150 | 8                                  | 0  | 5   | 2  | 9  | 214 |
| 0.1000 | 10                                 | 5  | 5   | 6  | 9  | 456 |

**Table 1-** Number of components of each feature selected for different C-values in LASSO.

To investigate optimal C-value to perform feature selection, we explore the class accuracy for different C-values. The logistic regression in L1 regularisation performs one-VS-rest classification, selecting best C-value by class. Through cross-validation,  $C = 0.002$  was selected as optimal C-value, which corresponds to only HOG columns (Table 1), indicating that this C-value may have been chosen as a local best based on a single family of class's CV performance. Further investigating using a plot of local accuracy vs. C, we found  $\text{logreg\_cv\_C\_}$  to be non-uniform across classes (Figure 3).

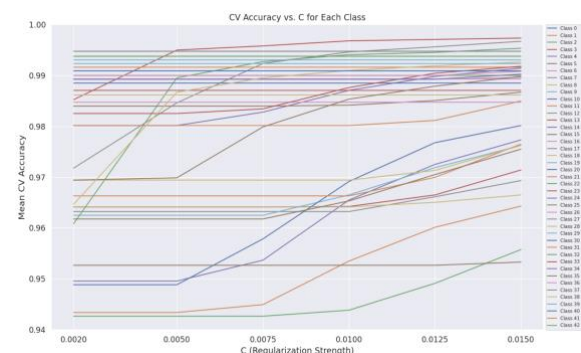


**Figure 3-** Local Accuracy vs. C for Each Class

While some classes' accuracy plateaued across C-values, others steadily increased, making 0.015 the global best C-value. This is supported by an analysis of global accuracy (Figure 4).

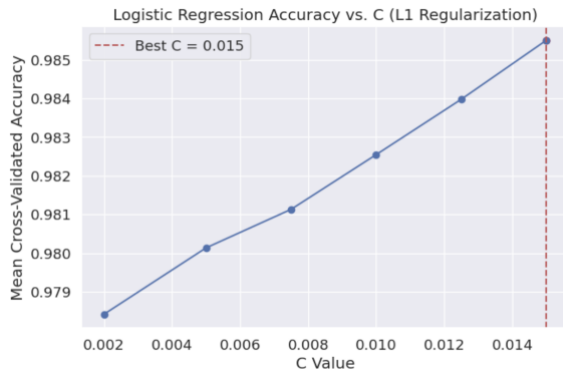


Figure 4- Global Accuracy vs. C for Each Class

In consideration of both the local and global best C-values, we selected  $C = 0.01$ , which provided a trade-off between generalisation and feature richness, selecting 150 features across a diverse set of features (LBP, DCT, GF, HOG), allowing models to leverage different visual representations and achieve compactness.

### 2.5.1 Feature Enhancement (deep learning)

Various feature enhancement techniques were applied to training set. Histogram equalization (HE) increased image contrast. Contrast Limited Adaptive Histogram Equalization (CLAHE) splits the image into tiles, applies HE to those tiles individually instead of globally, limiting the contrast enhancement to limit amplification of noise. The effect of applying CLAHE to original and augmented image for `img_000791.jpg` is shown in Figure 5.

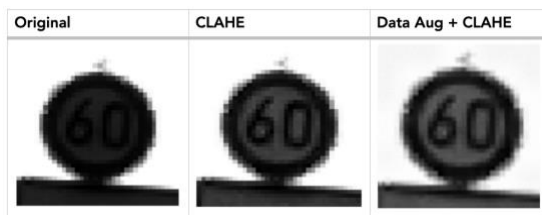


Figure 5- Images with different preprocessing.

## 2.6 Classifier Training

We trained a variety of classifiers to evaluate the effectiveness of our features and to identify the best-performing model. Each classical ML method was trained with **stratified 5-fold cross validation, tuned with grid search** to ensure optimal and consistent evaluation across dataset. To ensure distance measures (for KNN, SVM) remained meaningful, and that no data leakage occurred, features were standardised after train-validate split.

### 2.6.1 Random Forest Classifier

| RF Hyper-parameters |  |
|---------------------|--|
|---------------------|--|

|                   |      |
|-------------------|------|
| max_depth         | none |
| min_samples_split | 2    |
| n_estimators      | 200  |

Table 2- RF Hyper-parameters

Observe Table 2 for **RF**, `max_depth = none` indicates that the model benefits from capturing complex patterns in the data and affirms that the given selected features were informative enough. Furthermore, `min_samples_split = 2` implies that each tree is permitted to split to its ends, creating fine decision boundaries which could contribute to some overfitting. With `n_estimators = 200`, the predictions stabilised from `n_estimators = 100` due to reduction in variance and mitigates some effects of overfitting from lack of `max_depth` limit and minimal `min_samples_split`.

### 2.6.2 K-Nearest Neighbours Classifier

| KNN Hyper-parameters |   |
|----------------------|---|
| n_neighbors          | 1 |

Table 3- KNN Hyper-parameters

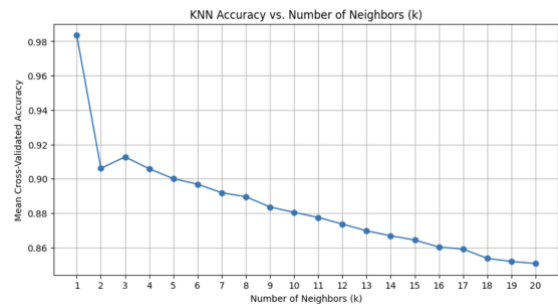


Figure 6- K-Nearest Neighbours n\_neighbors

Observe Figure 6 for **KNN**, `n_neighbors = 1` (Table 3) stood out exceptionally, producing an overall decreasing trend for validation accuracy as `n_neighbors` increased. This could be caused by class imbalance, where some classes have very limited instances (e.g. class 19 having only 29 instances) leading to a higher likelihood of misclassification when exposed to a greater number of neighbours. It is expected that `n_neighbors` could increase with more training data to help reduce overfitting due to noise.

### 2.6.3 Support Vector Machine Classifier

| SVM Hyper-parameters |       |
|----------------------|-------|
| C                    | 10    |
| gamma                | scale |
| kernel               | rbf   |

Table 4- SVM Hyper-parameters

Observe Table 4 for **SVM**, the C value of 10 indicates heavy penalisation for misclassified validation instances in the hyperplane fitting process. With data augmentation supporting better generalisation of the models, SVM can afford making a tighter decision boundary. The gamma remained default, which indicates the adaptability provided by scale was suitable to support data generalisability and stability. Similarly, the default kernel function rbf worked well in mapping the non-linear features and capturing complex curved relationships. This supports SVM's high performance as it is used in conjunction with HOG and LBP, two features which are highly dimensional.

## 2.6.4 Stacking Ensemble Classifier

**Stacking** combined the strengths of three base models using the XGBoost metaclassifier, chosen for its computational efficiency and built-in regularization to prevent overfitting. KNN being able to effectively classify class in the k=1 neighbourhood indicates that it has learnt to cluster similar classes (e.g. with shape or texture consistency) in a high-dimensional feature space. Where KNN performance is affected by the curse of dimensionality (distance losing interpretability), SVM can fill the gaps with regularisation, generalising on unseen data. On the other hand, RF was able to contribute to general robustness, catching global patterns that may not be as fine-grained with clear boundaries.

## 2.6.5 Convolutional Neural Network

We implemented our **CNN** model using the VGG16 architecture (Simonyan & Zisserman, 2014), which consists of 13 3x3 kernel-sized convolutional layers (2 layers of 64 filters, 2 layers of 128 filters, 3 layers of 256 filters, 6 layers of 512 layers), 3 dense layers and 5 Max Pooling layers. (Figure 7)

The main difference between the original architecture and our implementation is the usage of 128x128 tensor size (image size) instead of 224x224, to speed up training time. We also changed the number of dense neurons to 43, to match the number of traffic sign classes. We used early stopping to stop model training when validation loss stops decreasing, as well as model checkpoints to save the best model at each epoch. Each layer uses L2 regularization to avoid overfitting, without reducing the weights of less important features to 0. Random dropout was also applied before

the final dense layer. Each layer uses the ReLU activation function. To maintain a balance of training time and performance (Figure 13), we trained with batch size = 32 and 40 epochs.

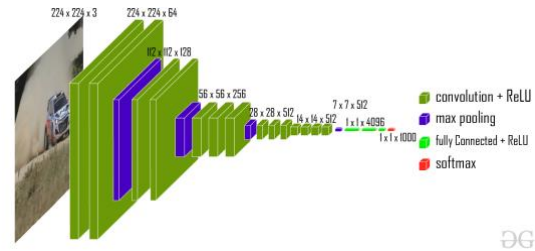


Figure 7- VGG16 architecture. Adapted from GeeksforGeeks.

| CNN Optimiser Hyper-parameters |      |
|--------------------------------|------|
| epochs                         | 40   |
| batch size                     | 32   |
| epsilon                        | 1e-7 |
| learning_rate                  | 1e-4 |

Table 5- CNN Optimiser Hyper-parameters

For **CNN**, we used the Adam optimizer since it requires minimal hyperparameter tuning and uses momentum, which speeds up gradient descent. Observe Table 5, epsilon is an additional variable used in the optimizer to prevent division by 0. By convention, it is set to a very small value to avoid significant effects on performance. As for learning rate, it controls the size of the step the Adam optimizer takes when doing gradient descent. To stabilize gradient descent and avoid overfitting, 1e-4 was selected (after previously training with larger steps such as 1e-3). This, however, slows down training.

## 3. Results

This section outlines the performance of each of our classifiers on 5-fold stratified cross validation. The results clearly show that CNN is the best classifier for this task, with the highest accuracy, precision and recall. (Table 6)

| Model        | 42% Test Accuracy | Validation Accuracy | Weighted Precision | Weighted Recall |
|--------------|-------------------|---------------------|--------------------|-----------------|
| <b>RF</b>    | 0.86486           | 0.9575              | 0.96               | 0.96            |
| <b>KNN</b>   | 0.88488           | 0.9793              | 0.98               | 0.98            |
| <b>SVM</b>   | 0.95295           | 0.9852              | 0.99               | 0.99            |
| <b>Stack</b> | 0.94094           | 0.9736              | 0.97               | 0.97            |
| <b>CNN</b>   | 0.99499           | 0.98998             | 0.99               | 0.99            |

Table 6- Evaluation metrics for different models.

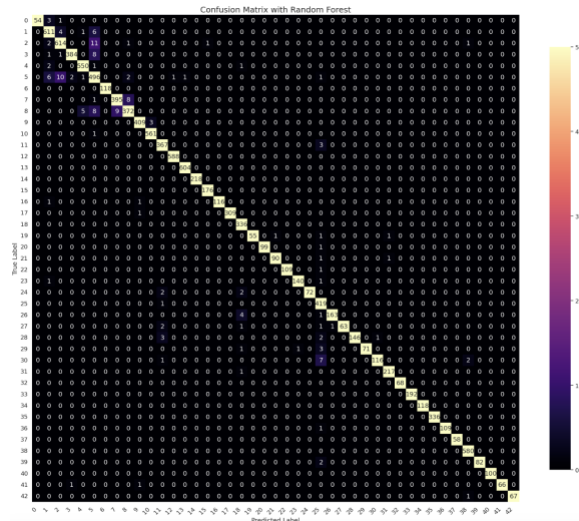


Figure 8- Confusion Matrix for RF.

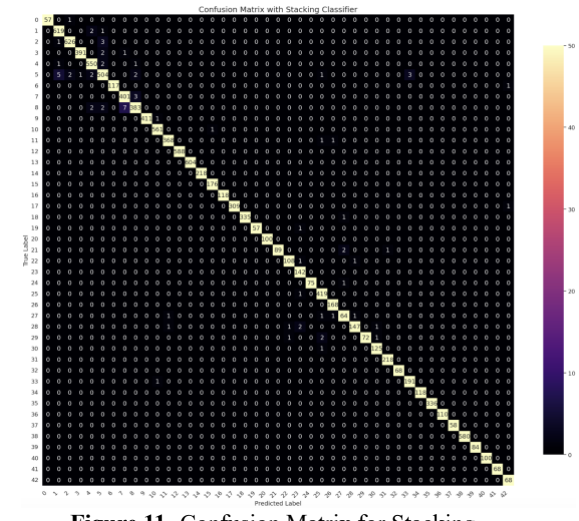


Figure 11- Confusion Matrix for Stacking.

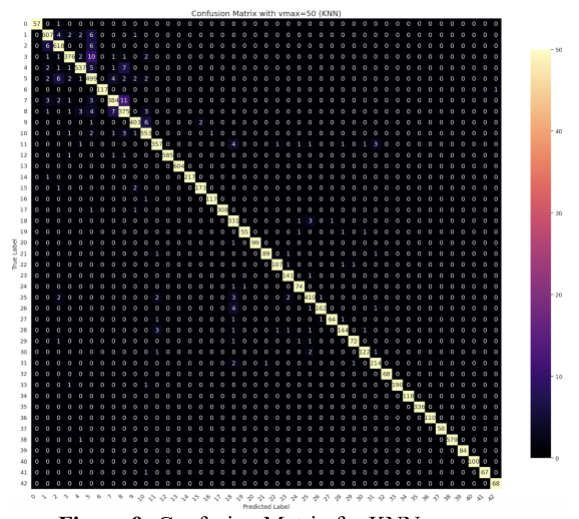


Figure 9- Confusion Matrix for KNN.

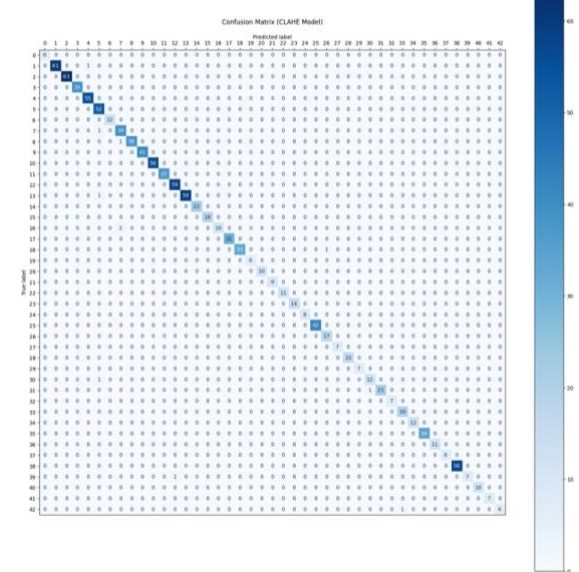


Figure 12- Confusion Matrix for CNN.

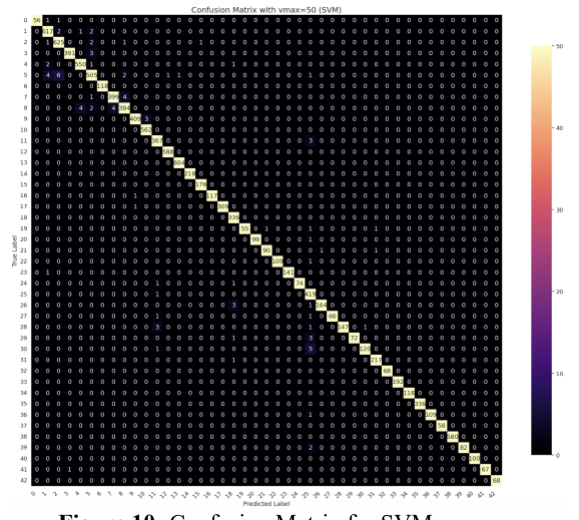


Figure 10- Confusion Matrix for SVM.

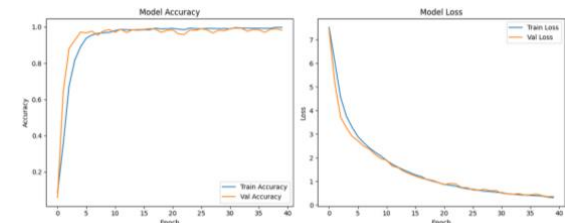


Figure 13- Model Accuracy and loss for KNN.

## 4. Discussion and Critical Analysis

### 4.1 Performance Interpretation

RF provided a robust model which helped capture feature interactions. However, it struggled with some visually similar classes as per confusion matrix illustrated (Figure 8). While it was able to capture global features to achieve that consistency, the RF model

performed worse for visually similar classes (Table 7) sharing the same shapes and colours which required more subtle feature analysis. This was reflected in the 42% test set where there was a notable (9.264%) accuracy drop.

| Predicted Class                                                                   |                                                                                   | Actual Class                                                                      |                                                                                   |                                                                                   |
|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
|  |  |  |  |  |
| 1                                                                                 | 2                                                                                 | 0                                                                                 | 3                                                                                 | 5                                                                                 |
|  |  |  |  |  |
| 11                                                                                | 25                                                                                | 28                                                                                | 29                                                                                | 30                                                                                |

Table 7- Misclassification of visually similar classes.

KNN performed well on local decision boundaries, but its dependence on distance metrics and low `n_neighbors` value of 1 led to the model being prone to overfitting. As such, we expected KNN to perform worse than other models despite high validation score. This was confirmed in 42% test set with an accuracy drop of 9.442% from the validation set (Table 6).

SVM performed best amongst our classic ML models. However, eight of its misclassified instances were between classes 7 and 8 (Table 8), suggesting an improvement from RF and KNN in providing a model that was able to classify many nuanced differences between very similar classes, but limitations still exist.

| Class 7                                                                             | Class 8                                                                             |
|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
|  |  |

Table 8- Misclassification of visually similar classes.

We expected **stacking** to perform better than its base models but turned out to be worse than its base model (SVM) by 1.201% in 42% test (Table 6). This could be attributed to the ensemble mechanism in stacking, which tries to select the best base classifier for each individual class. However, confusion matrices of RF, KNN, and SVM (Figure 8, 9, 10) indicates that their misclassifications largely belonged to the same class, meaning the best base classifier dominated.

CNN performed the best overall. This was expected as CNN excel at image recognition due to its ability to learn relevant features within the model, leading to effective learning and high generalisability. Upon investigating

the 9 of the 11 instances which it misclassified, we can infer that the low quality of images likely contributed to the model's inability to extract useful features from them (Table 9).

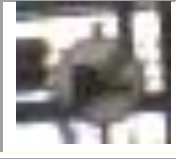
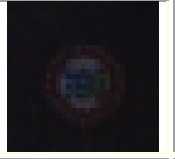
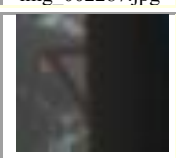
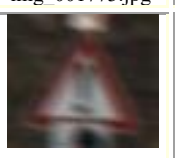
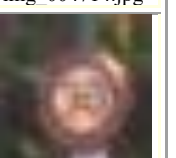
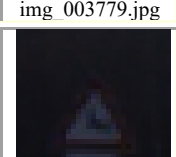
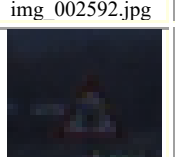
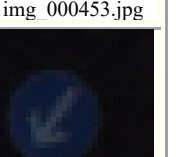
|                                                                                    |                                                                                     |                                                                                     |
|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
|  |  |  |
|  |  |  |
|  |  |  |

Table 9- 9 of 11 Misclassified instances in CNN.

## 4.2 Bias and Variance Analysis

Bias existing in models could be seen from consistent misclassification for a particular class, while variance could be deduced if a notable drop in accuracy for 42% test compared to validation is observed, suggesting limited model generalisability.

RF showed high variance, which could be attributed to its hyper-parameters `max_depth = None` and `min_samples_split = 2`, leading to some degree of overfitting on the training set.

KNN showed high variance. Since KNN retains 100% accuracy when classifying on development (training) instances, KNN is prone to memorises rather than learning. Additionally, the curse of dimensionality and low `n_neighbors` contributed to overfitting.

SVM showed relatively low variance due to regularisation and high `C` value, which decreased variance for better generalisability.

**Stacking** showed similar level of variance as SVM, however, it is expected that it will have slightly higher variance in the full testing set, as it also considers higher variance base models.

CNN exhibited a noticeable bias in 2 of its 11 misclassified instances. Table 10 shows two

instances of class 16, both of which were misclassified as class 7 by CNN. Given that it is a relatively clear image with little noise, we may attribute this error to the class imbalance of training data (59 instances of class 16, 202 instances of class 7, Figure 1). On the other hand, it is interesting that CNN was the only model whose validation accuracy was lower than its 42% test accuracy (Table 6). This suggests that CNN has better generalisability and lower variance compared to other models.



**Table 10-** 2 of 11 Misclassified instances in CNN.

### 4.3 Limitations

**Quality of data** was a significant limitation in this task. Most images were under 100x100, and many images presented difficulties (Table 11). The noisy data may have contributed to overfitting of models.



**Table 11-** Examples of difficult instances in training.

### 4.4 Trade-offs

**Data augmentation** presented significant trade-offs. While we were able to generate more instances to train on, low quality of training data often led to the generation of noisy data. RandomBrightnessContrast also rendered the colour histograms feature unusable.

The **classic ML models** each had their own trade-offs in hyper-parameters as explained throughout Section 4.1. As the models grew

more complex for the classification problem, it became more computationally expensive and traded in interpretability of model.

In training CNN, numerous choices led to trade-offs. Resizing the images to 128x128 resulted in higher quality images losing information, and lower quality images becoming harder to decipher. Furthermore, CNN has inherent randomness due to dropout, data augmentation, and image processing order by GPU), leading to some instability in performance.

## 5. Conclusion

A wide variety of preprocessing techniques, feature extraction, selection and enhancements were applied to improve models and compare their performance. While data augmentation increased training data diversity, it was insufficient in addressing class imbalances. Challenges such as long training times, inherent model randomness and poor data quality also proved to be significant obstacles.

The trained models each presented respective strengths and weaknesses. **Random Forest** was robust and capable of handling complex feature relations but lacked in stability of performance. **K-Nearest Neighbours** performed well on validation data but showed signs of overfitting. **Support Vector Machine** was stable, had low variance but made mistakes on more nuanced feature differences. **Stacking** underperformed relative to expectations due to suboptimal combination of base classifiers. In the end, **Convolutional Neural Networks** performed best as expected, with a high accuracy and low variance. Hence, it was chosen as the final model.

## 6. Reference

GeeksforGeeks. (n.d). *VGG-16 architecture*.  
GeeksforGeeks.

<https://www.geeksforgeeks.org/vgg-16-cnn-model/>

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. for vgg-16 architecture